SCIENTIFIC
AND
ENGINEERING
COMPUTATION
SERIES

PVM

Parallel Virtual Machine

A Users' Guide and Tutorial for Networked Parallel Computing

Al Geist
Adam Beguelin
Jack Dongarra
Weicheng Jiang
Robert Manchek
Vaidy Sunderam

Copyrighted Material
Downloaded from http://direct.mit.edu/books/monograph/2089/bookpreview-pdf/2430723 by guest on 25 May 2025

C Interface

```
#include "pvm3.h"
```

Process Control

```
int tid  = pvm_mytid(void)
int info = pvm_exit( void)
int info = pvm_kill( int tid)
int info = pvm_addhosts(char **hosts, int nhost,
                        int *infos)
int info = pvm_delhosts(char **hosts, int nhost,
                        int *infos)
int numt = pvm_spawn(char *task, char **argv,
                     int flag, char *where,
                     int ntask, int *tids)
```

Spawn flag options		MEANING
PvmTaskDefault	0	don't care where
PvmTaskHost	1	"where" contains host
PvmTaskArch	2	"where" contains arch
PvmHostCompl	32	use complement host set

Information

```
int tid  = pvm_parent(void)
int dtid = pvm_tidtohost(int tid)
int info = pvm_perror(char *msg)
int info = pvm_config(int *nhost, int *narch,
                      struct pvmhostinfo **hostp)
int info = pvm_tasks(int which, int *ntask,
                     struct pvmtaskinfo **taskp)
int    val = pvm_getopt(int what)
int oldval = pvm_setopt(int what, int val)
```

what option		SETS/GETS This
PvmRoute	1	routing policy : PvmRouteDirect
PvmDebugMask	2	debug level — PvmAllowDirect
more...	*	see man page

Signalling

```
int info = pvm_sendsig(int tid, int signum)
int info = pvm_notify(int about, int msgtag,
                      int ntask, int *tids)
```

About options		MEANING
PvmTaskExit	1	notify if task exit
PvmHostDelete	2	notify if deletion
PvmHostAdd	3	notify if addition

Message Buffers

```
int bufid  = pvm_mkbuf(int encoding)
int info   = pvm_freebuf(int bufid)
int bufid  = pvm_getsbuf(void)
int bufid  = pvm_getrbuf(void)
int oldbuf = pvm_setsbuf(int bufid)
int oldbuf = pvm_setrbuf(int bufid)
int bufid  = pvm_initsend(int encoding)
```

Encoding options		MEANING
PvmDataDefault	0	XDR
PvmDataRaw	1	no encoding
PvmDataInPlace	2	data left in place

Sending

```
int info = pvm_packf(  printf-like format... )
int info   pvm_pkbyte( char   *cp, int cnt, int std)
int info   pvm_pkcplx( float  *xp, int cnt, int std)
int info   pvm_pkdcplx(double *zp, int cnt, int std)
int info   pvm_pkdouble(double *dp, int cnt, int std)
int info   pvm_pkfloat(float  *fp, int cnt, int std)
int info   pvm_pkint(  int    *np, int cnt, int std)
int info = pvm_pklong( long   *np, int cnt, int std)
int info = pvm_pkshort(short  *np, int cnt, int std)
int info = pvm_pkstr(  char   *cp)
int info = pvm_send( int  tid, int msgtag)
int info = pvm_mcast(int *tids, int ntask, int msgtag)
int info = pvm_psend( int tid, int msgtag,
               void *vp, int cnt, int type)
```

Receiving

```
int bufid = pvm_recv( int tid, int msgtag)
int bufid = pvm_probe(int tid, int msgtag)
int bufid = pvm_nrecv(int tid, int msgtag)
int bufid = pvm_precv(int tid, int msgtag,
                void *vp, int cnt, int type
                int *rtid, int *rtag, int *rlen)
int bufid = pvm_trecv(int tid, int msgtag,
                        struct timeval *tmout)
int info = pvm_bufinfo(int bufid, int *bytes,
                       int *msgtag, int *tid)
int info = pvm_unpackf(  printf-like format...)
int info = pvm_upkbyte( char   *cp, int cnt, int std)
int info = pvm_upkcplx( float  *xp, int cnt, int std)
int info = pvm_upkdcplx(double *zp, int cnt, int std)
int info = pvm_upkdouble(double *dp, int cnt, int std)
int info = pvm_upkfloat(float  *fp, int cnt, int std)
int info = pvm_upkint(  int    *np, int cnt, int std)
int info = pvm_upklong( long   *np, int cnt, int std)
int info = pvm_upkshort(short  *np, int cnt, int std)
int info = pvm_upkstr(  char   *cp)
```

Group Operations

```
int inum = pvm_joingroup(char *group)
int info = pvm_lvgroup( char *group)
int size = pvm_gsize(   char *group)
int tid  = pvm_gettid(  char *group, int inum)
int inum = pvm_getinst( char *group, int tid)
int info = pvm_barrier( char *group, int count)
int info = pvm_bcast(   char *group, int msgtag)
int info = pvm_reduce(void *op, void *vp, int cnt,
      int type, int msgtag, char *group, int root)
```

op options	vp type options		
PvmMax	PVM_BYTE	PVM_FLOAT	PVM_STR
PvmMin	PVM_SHORT	PVM_DOUBLE	PVM_UINT
PvmSum	PVM_INT	PVM_CPLX	PVM_USHORT
PvmProduct	PVM_LONG	PVM_DCPLX	PVM_ULONG

Copyrighted Material

Downloaded from http://direct.mit.edu/books/monograph/2089/bookpreview-pdf/2430723 by guest on 25 May 2025

PVM: Parallel Virtual Machine

Copyrighted Material

Downloaded from http://direct.mit.edu/books/monograph/2089/bookpreview-pdf/2430723 by guest on 25 May 2025

Scientific and Engineering Computation
Janusz Kowalik, Editor

Data-Parallel Programming on MIMD Computers
by Philip J. Hatcher and Michael J. Quinn, 1991

Unstructured Scientific Computation on Scalable Multiprocessors
edited by Piyush Mehrotra, Joel Saltz, and Robert Voigt, 1991

Parallel Computational Fluid Dynamics: Implementations and Results
edited by Horst D. Simon, 1992

Enterprise Integration Modeling: Proceedings of the First International Conference
edited by Charles J. Petrie, Jr., 1992

The High Performance Fortran Handbook
by Charles H. Koelbel, David B. Loveman, Robert S. Schreiber, Guy L. Steele Jr., and Mary E. Zosel, 1994

Using MPI: Portable Parallel Programming with the Message-Passing Interface
by William Gropp, Ewing Lusk, and Anthony Skjellum, 1994

PVM: Parallel Virtual Machine – A Users' Guide and Tutorial for Networked Parallel Computing
by Al Geist, Adam Beguelin, Jack Dongarra, Weicheng Jiang, Robert Manchek, and Vaidy Sunderam, 1994

Copyrighted Material

Downloaded from http://direct.mit.edu/books/monograph/2089/bookpreview-pdf/2430723 by guest on 25 May 2025

PVM: Parallel Virtual Machine
A Users' Guide and Tutorial for Networked Parallel Computing

Al Geist
Adam Beguelin
Jack Dongarra
Weicheng Jiang
Robert Manchek
Vaidy Sunderam

The MIT Press
Cambridge, Massachusetts
London, England

Copyrighted Material

Downloaded from http://direct.mit.edu/books/monograph/2089/bookpreview-pdf/2430723 by guest on 25 May 2025

Fifth printing, 2000

© 1994 Massachusetts Institute of Technology

All rights reserved. No part of this book may be reproduced in any form by any electronic or mechanical means (including photocopying, recording, or information storage and retrieval) without permission in writing from the publisher.

This book was set in LaTeX by the authors and was printed and bound in the United States of America.

Library of Congress Cataloging-in-Publication Data

PVM parallel virtual machine — a users' guide and tutorial for networked parallel computing / Al Geist ... [et al].
p. cm. — (Scientific and engineering computation)
Includes bibliographical references and index.
ISBN 0-262-57108-0 (pbk. : acid-free paper)
1. Parallel computers. 2. Computer networks. I. Geist, Al. II. Series.
QA76.58.P85 1994
005.2—dc20 94-23404
CIP

This book is also available in postscript and html forms over the Internet.
To retrieve the postscript file you can use one of the following methods:

- anonymous ftp
  `ftp netlib2.cs.utk.edu`
  `cd pvm3/book`
  `get pvm-book.ps`
  `quit`
- from any machine on the Internet type:
  `rcp anon@netlib2.cs.utk.edu:pvm3/book/pvm-book.ps pvm-book.ps`
- sending email to `netlib@ornl.gov` and in the message type:
  `send pvm-book.ps from pvm3/book`
- use Xnetlib and click "library", click "pvm3", click "book", click "pvm3/pvm-book.ps", click "download", click "Get Files Now". (Xnetlib is an X-window interface to the netlib software based on a client-server model. The software can be found in netlib, "send index from xnetlib").

To view the html file use the URL:

- `http://www.netlib.org/pvm3/book/pvm-book.html`

Copyright ed Material

Downloaded from http://direct.mit.edu/books/monograph/2089/bookpreview-pdf/2430723 by guest on 25 May 2025

Contents

Series Foreword xi

Preface xiii

1 Introduction 1

1.1 Heterogeneous Network Computing 2

1.2 Trends in Distributed Computing 3

1.3 PVM Overview 5

1.4 Other Packages 6

1.4.1 The p4 System 6

1.4.2 Express 7

1.4.3 MPI 7

1.4.4 The Linda System 8

2 The PVM System 11

3 Using PVM 19

3.1 How to Obtain the PVM Software 19

3.2 Setup to Use PVM 19

3.3 Setup Summary 20

3.4 Starting PVM 22

3.5 Common Startup Problems 24

3.6 Running PVM Programs 25

3.7 PVM Console Details 27

3.8 Host File Options 29

4 Basic Programming Techniques 33

4.1 Common Parallel Programming Paradigms 33

4.1.1 Crowd Computations 34

4.1.2 Tree Computations 37

4.2 Workload Allocation 39

4.2.1 Data Decomposition 40

4.2.2 Function Decomposition 41

Copyrighted Material

Downloaded from http://direct.mit.edu/books/monograph/2089/bookpreview-pdf/2430723 by guest on 25 May 2025

4.3 Porting Existing Applications to PVM 43

5 PVM User Interface 45

5.1 Process Control 46
5.2 Information 49
5.3 Dynamic Configuration 50
5.4 Signaling 51
5.5 Setting and Getting Options 52
5.6 Message Passing 53
5.6.1 Message Buffers 53
5.6.2 Packing Data 55
5.6.3 Sending and Receiving Data 57
5.6.4 Unpacking Data 59
5.7 Dynamic Process Groups 60

6 Program Examples 63

6.1 Fork-Join 63
6.2 Dot Product 68
6.3 Failure 73
6.4 Matrix Multiply 76
6.5 One-Dimensional Heat Equation 83
6.5.1 Different Styles of Communication 91

7 How PVM Works 93

7.1 Components 93
7.1.1 Task Identifiers 93
7.1.2 Architecture Classes 95
7.1.3 Message Model 95
7.1.4 Asynchronous Notification 96
7.1.5 PVM Daemon and Programming Library 96
7.2 Messages 97
7.2.1 Fragments and Databufs 97

Copyrighted Material

Downloaded from http://direct.mit.edu/books/monograph/2089/bookpreview-pdf/2430723 by guest on 25 May 2025

7.2.2 Messages in Libpvm 97
7.2.3 Messages in the Pvmd 98
7.2.4 Pvmd Entry Points 98
7.2.5 Control Messages 101
7.3 PVM Daemon 101
7.3.1 Startup 101
7.3.2 Shutdown 102
7.3.3 Host Table and Machine Configuration 102
7.3.4 Tasks 104
7.3.5 Wait Contexts 105
7.3.6 Fault Detection and Recovery 106
7.3.7 Pvmd' 106
7.3.8 Starting Slave Pvmds 107
7.3.9 Resource Manager 109
7.4 Libpvm Library 111
7.4.1 Language Support 111
7.4.2 Connecting to the Pvmd 111
7.5 Protocols 112
7.5.1 Messages 112
7.5.2 Pvmd-Pvmd 112
7.5.3 Pvmd-Task and Task-Task 116
7.6 Message Routing 117
7.6.1 Pvmd 117
7.6.2 Pvmd and Foreign Tasks 118
7.6.3 Libpvm 119
7.6.4 Multicasting 120
7.7 Task Environment 121
7.7.1 Environment Variables 121
7.7.2 Standard Input and Output 122
7.7.3 Tracing 124
7.7.4 Debugging 124
7.8 Console Program 124
7.9 Resource Limitations 125

Copyrighted Material

Downloaded from http://direct.mit.edu/books/monograph/2089/bookpreview-pdf/2430723 by guest on 25 May 2025

7.9.1 In the PVM Daemon 125
7.9.2 In the Task 126
7.10 Multiprocessor Systems 126
7.10.1 Message-Passing Architectures 127
7.10.2 Shared-Memory Architectures 130
7.10.3 Optimized Send and Receive on MPP 132

8 Advanced Topics 135

8.1 XPVM 135
8.1.1 Network View 137
8.1.2 Space-Time View 138
8.1.3 Other Views 139
8.2 Porting PVM to New Architectures 139
8.2.1 Unix Workstations 140
8.2.2 Multiprocessors 142

9 Troubleshooting 147

9.1 Getting PVM Installed 147
9.1.1 Set PVM_ROOT 147
9.1.2 On-Line Manual Pages 148
9.1.3 Building the Release 148
9.1.4 Errors During Build 148
9.1.5 Compatible Versions 149
9.2 Getting PVM Running 149
9.2.1 Pvmd Log File 149
9.2.2 Pvmd Socket Address File 150
9.2.3 Starting PVM from the Console 150
9.2.4 Starting the Pvmd by Hand 151
9.2.5 Adding Hosts to the Virtual Machine 151
9.2.6 PVM Host File 152
9.2.7 Shutting Down 152
9.3 Compiling Applications 153
9.3.1 Header Files 153

Copyrighted Material

Downloaded from http://direct.mit.edu/books/monograph/2089/bookpreview-pdf/2430723 by guest on 25 May 2025

9.3.2 Linking 153

9.4 Running Applications 154

9.4.1 Spawn Can't Find Executables 154

9.4.2 Group Functions 154

9.4.3 Memory Use 154

9.4.4 Input and Output 155

9.4.5 Scheduling Priority 156

9.4.6 Resource Limitations 157

9.5 Debugging and Tracing 157

9.6 Debugging the System 158

9.6.1 Runtime Debug Masks 159

9.6.2 Tickle the Pvmd 159

9.6.3 Starting Pvmd under a Debugger 160

9.6.4 Sane Heap 160

9.6.5 Statistics 161

Glossary 163

A History of PVM Versions 175

B PVM 3 Routines 181

Bibliography 275

Index 277

Copyrighted Material

Downloaded from http://direct.mit.edu/books/monograph/2089/bookpreview-pdf/2430723 by guest on 25 May 2025

Copyrighted Material

Downloaded from http://direct.mit.edu/books/monograph/2089/bookpreview-pdf/2430723 by guest on 25 May 2025

Series Foreword

The world of modern computing potentially offers many helpful methods and tools to scientists and engineers, but the fast pace of change in computer hardware, software, and algorithms often makes practical use of the newest computing technology difficult. The Scientific and Engineering Computation series focuses on rapid advances in computing technologies and attempts to facilitate transferring these technologies to applications in science and engineering. It will include books on theories, methods, and original applications in such areas as parallelism, large-scale simulations, time-critical computing, computer-aided design and engineering, use of computers in manufacturing, visualization of scientific data, and human-machine interface technology.

The series will help scientists and engineers to understand the current world of advanced computation and to anticipate future developments that will impact their computing environments and open up new capabilities and modes of computation.

This volume presents a software package for developing parallel programs executable on networked Unix computers. The tool called Parallel Virtual Machine (PVM) allows a heterogeneous collection of workstations and supercomputers to function as a single high-performance parallel machine. PVM is portable and runs on a wide variety of modern platforms. It has been well accepted by the global computing community and used successfully for solving large-scale problems in science, industry, and business.

Janusz S. Kowalik

Copyrighted Material

Downloaded from http://direct.mit.edu/books/monograph/2089/bookpreview-pdf/2430723 by guest on 25 May 2025

Copyrighted Material

Downloaded from http://direct.mit.edu/books/monograph/2089/bookpreview-pdf/2430723 by guest on 25 May 2025

Preface

In this book we describe the Parallel Virtual Machine (PVM) system and how to develop programs using PVM. PVM is a software system that permits a heterogeneous collection of Unix computers networked together to be viewed by a user's program as a single parallel computer. PVM is the mainstay of the Heterogeneous Network Computing research project, a collaborative venture between Oak Ridge National Laboratory, the University of Tennessee, Emory University, and Carnegie Mellon University.

The PVM system has evolved in the past several years into a viable technology for distributed and parallel processing in a variety of disciplines. PVM supports a straightforward but functionally complete message-passing model.

PVM is designed to link computing resources and provide users with a parallel platform for running their computer applications, irrespective of the number of different computers they use and where the computers are located. When PVM is correctly installed, it is capable of harnessing the combined resources of typically heterogeneous networked computing platforms to deliver high levels of performance and functionality.

In this book, we describe the architecture of the PVM system and discuss its computing model; the programming interface it supports; auxiliary facilities for process groups; the use of PVM on highly parallel systems such as the Intel Paragon, Cray T3D, and Thinking Machines CM-5; and some of the internal implementation techniques employed. Performance issues, dealing primarily with communication overheads, are analyzed, and recent findings as well as enhancements are presented. To demonstrate the viability of PVM for large-scale scientific supercomputing, we also provide some example programs.

This book is not a textbook; rather, it is meant to provide a fast entrance to the world of heterogeneous network computing. We intend this book to be used by two groups of readers: students and researchers working with networks of computers. As such, we hope this book can serve both as a reference and as a supplement to a teaching text on aspects of network computing.

This guide will familiarize readers with the basics of PVM and the concepts used in programming on a network. The information provided here will help with the following PVM tasks:

- Writing a program in PVM
- Building PVM on a system
- Starting PVM on a set of machines
- Debugging a PVM application

Copyrighted Material

Downloaded from http://direct.mit.edu/books/monograph/2089/bookpreview-pdf/2430723 by guest on 25 May 2025

A Bit of History

The PVM project began in the summer of 1989 at Oak Ridge National Laboratory. The prototype system, PVM 1.0, was constructed by Vaidy Sunderam and Al Geist; this version of the system was used internally at the Lab and was not released to the outside. Version 2 of PVM was written at the University of Tennessee and released in March 1991. During the following year, PVM began to be used in many scientific applications. After user feedback and a number of changes (PVM 2.1 - 2.4), a complete rewrite was undertaken, and version 3 was completed in February 1993. It is PVM version 3.3 that we describe in this book (and refer to simply as PVM). The PVM software has been distributed freely and is being used in computational applications around the world.

Who Should Read This Book?

To successfully use this book, one should be experienced with common programming techniques and understand some basic parallel processing concepts. In particular, this guide assumes that the user knows how to write, execute, and debug Fortran or C programs and is familiar with Unix.

Typographical Conventions

We use the following conventions in this book:

- Terms used for the first time, variables, and book titles are in *italic type*. For example: For further information on *PVM daemon* see the description in *PVM: Parallel Virtual Machine – A Users' Guide and Tutorial for Networked Parallel Computing.*
- Text that the user types is in **`Courier bold font`**. For example: **`$ pvm`**

The Map

This guide is divided into three major parts; it includes nine chapters, a glossary, two appendixes and a bibliography.

- Part I – Basics (Chapters 1–6). This part provides the facts, as well as some interpretation of the underlying system. It describes the overall concepts, system, and techniques for making PVM work for applications.
  - Introduction to PVM – introduction to network computing and PVM; terms and concepts, including an overview of the system

Copyrighted Material

Downloaded from http://direct.mit.edu/books/monograph/2089/bookpreview-pdf/2430723 by guest on 25 May 2025

- Overview of PVM
  * C, C++, and Fortran
  * basic principles
  * "hello.c" example
  * other systems (e.g., MPI)
- PVM Tutorial
  * setting up PVM
  * running an existing program
  * console
  * XPVM
- Programming
  * basic programming techniques
  * data decomposition / partitioning
  * function decomposition
  * putting PVM in existing code
- User Interface
  * functions
  * hostfile
- Program Examples
  * PVM programs

• Part 2 – Details (Chapters 7–9). This part describes the internals of PVM.

- How PVM Works
  * Unix hooks to PVM interfaces
  * multiprocessors - shared and distributed memory
- Advanced Topics
  * portability
  * debugging
  * tracing
  * XPVM details
- Troubleshooting; interesting tidbits and information on PVM, including frequently asked questions.

Copyrighted Material

Downloaded from http://direct.mit.edu/books/monograph/2089/bookpreview-pdf/2430723 by guest on 25 May 2025

- Part 3 – The Remains. This part provides some useful information on the use of the PVM interface.
  - Glossary of Terms: gives a short description for terms used in the PVM context.
  - Appendix A, History of PVM versions: list of all the versions of PVM that have been released from the first one in February 1991 through July 1994. Along with each version we include a brief synopsis of the improvements made in version 3.
  - Appendix B, Man Pages: provides an alphabetical listing of all the PVM 3 routines. Each routine is described in detail for both C and Fortran use. There are examples and diagnostics for each routine.
  - Quick Reference Card for PVM: provides the name and calling sequence for the PVM routines in both C and Fortran. (If this card is missing from the text a replacement can be downloaded over the network by ftp'ing to `netlib2.cs.utk.edu`; `cd pvm3/book`; `get refcard.ps`.)
  - Bibliography

Comments and Questions

PVM is an ongoing research project. As such, we provide limited support. We welcome feedback on this book and other aspects of the system to help in enhancing PVM. Please send comments and questions to `pvm@msr.epm.ornl.gov`. by e-mail. While we would like to respond to all the electronic mail received, this may not be always possible. We therefore recommend also posting messages to the newsgroup `comp.parallel.pvm`. This unmoderated newsgroup was established on the Internet in May 1993 to provide a forum for discussing issues related to the use of PVM. Questions (from beginner to the very experienced), advice, exchange of public-domain extensions to PVM, and bug reports can be posted to the newsgroup.

Acknowledgments

We gratefully acknowledge the valuable assistance of many people who have contributed to the PVM project. In particular, we thank Peter Rigsbee and Neil Lincoln for their help and insightful comments. We thank the PVM group at the University of Tennessee and Oak Ridge National Laboratory—Carolyn Aebischer, Martin Do, June Donato, Jim Kohl, Keith Moore, Phil Papadopoulos, and Honbo Zhou—for their assistance with the development of various pieces and components of PVM. In addition we express appreciation to all those who helped in the preparation of this work, in particular to Clint

Copyrighted Material

Downloaded from http://direct.mit.edu/books/monograph/2089/bookpreview-pdf/2430723 by guest on 25 May 2025

Whaley and Robert Seccomb for help on the examples, Ken Hawick for contributions to the glossary, and Gail Pieper for helping with the task of editing the manuscript.

A number of computer vendors have encouraged and provided valuable suggestions during the development of PVM. We thank Cray Research Inc., IBM, Convex Computer, Silicon Graphics, Sequent Computer, and Sun Microsystems for their assistance in porting the software to their platforms.

This work would not have been possible without the support of the Office of Scientific Computing, U.S. Department of Energy, under Contract DE-AC05-84OR21400; the National Science Foundation Science and Technology Center Cooperative Agreement No. CCR-8809615; and the Science Alliance, a state-supported program at the University of Tennessee.

Copyrighted Material

Downloaded from http://direct.mit.edu/books/monograph/2089/bookpreview-pdf/2430723 by guest on 25 May 2025

Copyrighted Material

Downloaded from http://direct.mit.edu/books/monograph/2089/bookpreview-pdf/2430723 by guest on 25 May 2025

PVM: Parallel Virtual Machine

Copyrighted Material

Downloaded from http://direct.mit.edu/books/monograph/2089/bookpreview-pdf/2430723 by guest on 25 May 2025

Copyrighted Material

Downloaded from http://direct.mit.edu/books/monograph/2089/bookpreview-pdf/2430723 by guest on 25 May 2025

1 Introduction

Parallel processing, the method of having many small tasks solve one large problem, has emerged as a key enabling technology in modern computing. The past several years have witnessed an ever-increasing acceptance and adoption of parallel processing, both for high-performance scientific computing and for more "general-purpose" applications, was a result of the demand for higher performance, lower cost, and sustained productivity. The acceptance has been facilitated by two major developments: massively parallel processors (MPPs) and the widespread use of distributed computing.

MPPs are now the most powerful computers in the world. These machines combine a few hundred to a few thousand CPUs in a single large cabinet connected to hundreds of gigabytes of memory. MPPs offer enormous computational power and are used to solve computational Grand Challenge problems such as global climate modeling and drug design. As simulations become more realistic, the computational power required to produce them grows rapidly. Thus, researchers on the cutting edge turn to MPPs and parallel processing in order to get the most computational power possible.

The second major development affecting scientific problem solving is *distributed computing.* Distributed computing is a process whereby a set of computers connected by a network are used collectively to solve a single large problem. As more and more organizations have high-speed local area networks interconnecting many general-purpose workstations, the combined computational resources may exceed the power of a single high-performance computer. In some cases, several MPPs have been combined using distributed computing to produce unequaled computational power.

The most important factor in distributed computing is cost. Large MPPs typically cost more than $10 million. In contrast, users see very little cost in running their problems on a local set of existing computers. It is uncommon for distributed-computing users to realize the raw computational power of a large MPP, but they are able to solve problems several times larger than they could using one of their local computers.

Common between distributed computing and MPP is the notion of message passing. In all parallel processing, data must be exchanged between cooperating tasks. Several paradigms have been tried including shared memory, parallelizing compilers, and message passing. The message-passing model has become the paradigm of choice, from the perspective of the number and variety of multiprocessors that support it, as well as in terms of applications, languages, and software systems that use it.

The Parallel Virtual Machine (PVM) system described in this book uses the message-passing model to allow programmers to exploit distributed computing across a wide variety of computer types, including MPPs. A key concept in PVM is that it makes a collection of computers appear as one large *virtual* machine, hence its name.

Copyrighted Material

Downloaded from http://direct.mit.edu/books/monograph/2089/bookpreview-pdf/2430723 by guest on 25 May 2025

1.1 Heterogeneous Network Computing

In an MPP, every processor is exactly like every other in capability, resources, software, and communication speed. Not so on a network. The computers available on a network may be made by different vendors or have different compilers. Indeed, when a programmer wishes to exploit a collection of networked computers, he may have to contend with several different types of heterogeneity:

- architecture,
- data format,
- computational speed,
- machine load, and
- network load.

The set of computers available can include a wide range of architecture types such as 386/486 PC class machines, high-performance workstations, shared-memory multiprocessors, vector supercomputers, and even large MPPs. Each architecture type has its own optimal programming method. In addition, a user can be faced with a hierarchy of programming decisions. The parallel virtual machine may itself be composed of parallel computers. Even when the architectures are only serial workstations, there is still the problem of incompatible binary formats and the need to compile a parallel task on each different machine.

Data formats on different computers are often incompatible. This incompatibility is an important point in distributed computing because data sent from one computer may be unreadable on the receiving computer. Message-passing packages developed for heterogeneous environments must make sure all the computers understand the exchanged data. Unfortunately, the early message-passing systems developed for specific MPPs are not amenable to distributed computing because they do not include enough information in the message to encode or decode it for any other computer.

Even if the set of computers are all workstations with the same data format, there is still heterogeneity due to different computational speeds. As an simple example, consider the problem of running parallel tasks on a virtual machine that is composed of one supercomputer and one workstation. The programmer must be careful that the supercomputer doesn't sit idle waiting for the next data from the workstation before continuing. The problem of computational speeds can be very subtle. The virtual machine can be composed of a set of identical workstations. But since networked computers can have several other users on them running a variety of jobs, the machine load can vary dramatically.

Copyrighted Material

Downloaded from http://direct.mit.edu/books/monograph/2089/bookpreview-pdf/2430723 by guest on 25 May 2025

The result is that the effective computational power across identical workstations can vary by an order of magnitude.

Like machine load, the time it takes to send a message over the network can vary depending on the network load imposed by all the other network users, who may not even be using any of the computers in the virtual machine. This sending time becomes important when a task is sitting idle waiting for a message, and it is even more important when the parallel algorithm is sensitive to message arrival time. Thus, in distributed computing, heterogeneity can appear dynamically in even simple setups.

Despite these numerous difficulties caused by heterogeneity, distributed computing offers many advantages:

- By using existing hardware, the cost of this computing can be very low.
- Performance can be optimized by assigning each individual task to the most appropriate architecture.
- One can exploit the heterogeneous nature of a computation. Heterogeneous network computing is not just a local area network connecting workstations together. For example, it provides access to different data bases or to special processors for those parts of an application that can run only on a certain platform.
- The virtual computer resources can grow in stages and take advantage of the latest computational and network technologies.
- Program development can be enhanced by using a familiar environment. Programmers can use editors, compilers, and debuggers that are available on individual machines.
- The individual computers and workstations are usually stable, and substantial expertise in their use is readily available.
- User-level or program-level fault tolerance can be implemented with little effort either in the application or in the underlying operating system.
- Distributed computing can facilitate collaborative work.

All these factors translate into reduced development and debugging time, reduced contention for resources, reduced costs, and possibly more effective implementations of an application. It is these benefits that PVM seeks to exploit. From the beginning, the PVM software package was designed to make programming for a heterogeneous collection of machines straightforward.

1.2 Trends in Distributed Computing

Stand-alone workstations delivering several tens of millions of operations per second are commonplace, and continuing increases in power are predicted. When these computer

Copyrighted Material

Downloaded from http://direct.mit.edu/books/monograph/2089/bookpreview-pdf/2430723 by guest on 25 May 2025

systems are interconnected by an appropriate high-speed network, their combined computational power can be applied to solve a variety of computationally intensive applications. Indeed, network computing may even provide supercomputer-level computational power. Further, under the right circumstances, the network-based approach can be effective in coupling several similar multiprocessors, resulting in a configuration that might be economically and technically difficult to achieve with supercomputer hardware.

To be effective, distributed computing requires high communication speeds. In the past fifteen years or so, network speeds have increased by several orders of magnitude (see Figure 1.1).

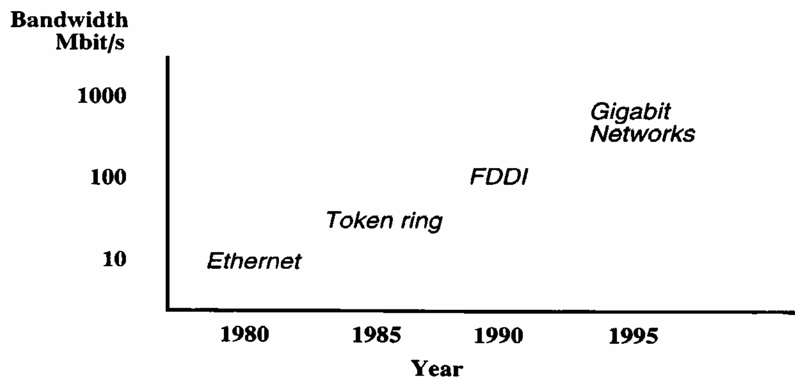

Figure 1.1
Networking speeds

Among the most notable advances in computer networking technology are the following:

- Ethernet – the name given to the popular local area packet-switched network technology invented by Xerox PARC. The Ethernet is a 10 Mbit/s broadcast bus technology with distributed access control.
- FDDI – the Fiber Distributed Data Interface. FDDI is a 100-Mbit/sec token-passing ring that uses optical fiber for transmission between stations and has dual counter-rotating rings to provide redundant data paths for reliability.
- HiPPI – the high-performance parallel interface. HiPPI is a copper-based data communications standard capable of transferring data at 800 Mbit/sec over 32 parallel lines or 1.6 Gbit/sec over 64 parallel lines. Most commercially available high-performance computers offer a HIPPI interface. It is a point-to-point channel that does not support multidrop configurations.

Copyrighted Material

Downloaded from http://direct.mit.edu/books/monograph/2089/bookpreview-pdf/2430723 by guest on 25 May 2025

- SONET – Synchronous Optical Network. SONET is a series of optical signals that are multiples of a basic signal rate of 51.84 Mbit/sec called OC-1. The OC-3 (155.52 Mbit/sec) and OC-12 (622.08 Mbit/sec) have been designated as the customer access rates in future B-ISDN networks, and signal rates of OC-192 (9.952 Gbit/sec) are defined.
- ATM – Asynchronous Transfer Mode. ATM is the technique for transport, multiplexing, and switching that provides a high degree of flexibility required by B-ISDN. ATM is a connection-oriented protocol employing fixed-size packets with a 5-byte header and 48 bytes of information.

These advances in high-speed networking promise high throughput with low latency and make it possible to utilize distributed computing for years to come. Consequently, increasing numbers of universities, government and industrial laboratories, and financial firms are turning to distributed computing to solve their computational problems. The objective of PVM is to enable these institutions to use distributed computing *efficiently*.

1.3 PVM Overview

The PVM software provides a unified framework within which parallel programs can be developed in an efficient and straightforward manner using existing hardware. PVM enables a collection of heterogeneous computer systems to be viewed as a single parallel virtual machine. PVM transparently handles all message routing, data conversion, and task scheduling across a network of incompatible computer architectures.

The PVM computing model is simple yet very general, and accommodates a wide variety of application program structures. The programming interface is deliberately straightforward, thus permitting simple program structures to be implemented in an intuitive manner. The user writes his application as a collection of cooperating *tasks*. Tasks access PVM resources through a library of standard interface routines. These routines allow the initiation and termination of tasks across the network as well as communication and synchronization between tasks. The PVM message-passing primitives are oriented towards heterogeneous operation, involving strongly typed constructs for buffering and transmission. Communication constructs include those for sending and receiving data structures as well as high-level primitives such as broadcast, barrier synchronization, and global sum.

PVM tasks may possess arbitrary control and dependency structures. In other words, at any point in the execution of a concurrent application, any task in existence may start or stop other tasks or add or delete computers from the virtual machine. Any process may communicate and/or synchronize with any other. Any specific control and dependency structure may be implemented under the PVM system by appropriate use

Copyrighted Material

Downloaded from http://direct.mit.edu/books/monograph/2089/bookpreview-pdf/2430723 by guest on 25 May 2025

of PVM constructs and host language control-flow statements.

Owing to its ubiquitous nature (specifically, the virtual machine concept) and also because of its simple but complete programming interface, the PVM system has gained widespread acceptance in the high-performance scientific computing community.

1.4 Other Packages

Several research groups have developed software packages that like PVM assist programmers in using distributed computing. Among the most well known efforts are P4 [1], Express [6], MPI [7], and Linda [3]. Various other systems with similar capabilities are also in existence; a reasonably comprehensive listing may be found in [17].

1.4.1 The p4 System

P4 [1] is a library of macros and subroutines developed at Argonne National Laboratory for programming a variety of parallel machines. The p4 system supports both the shared-memory model (based on monitors) and the distributed-memory model (using message-passing). For the shared-memory model of parallel computation, p4 provides a set of useful monitors as well as a set of primitives from which monitors can be constructed. For the distributed-memory model, p4 provides typed send and receive operations and creation of processes according to a text file describing group and process structure.

Process management in the p4 system is based on a configuration file that specifies the host pool, the object file to be executed on each machine, the number of processes to be started on each host (intended primarily for multiprocessor systems), and other auxiliary information. An example of a configuration file is

```
# start one slave on each of sun2 and sun3
local 0
sun2  1  /home/mylogin/p4pgms/sr_test
sun3  1  /home/mylogin/p4pgms/sr_test
```

Two issues are noteworthy in regard to the process management mechanism in p4. First, there is the notion a "master" process and "slave" processes, and multilevel hierarchies may be formed to implement what is termed a *cluster* model of computation. Second, the primary mode of process creation is static, via the configuration file; dynamic process creation is possible only by a statically created process that must invoke a special o4 function that spawns a new process on the local machine. Despite these restrictions, a variety of application paradigms may be implemented in the p4 system in a fairly straightforward manner.

Copyrighted Material

Downloaded from http://direct.mit.edu/books/monograph/2089/bookpreview-pdf/2430723 by guest on 25 May 2025

Message passing in the p4 system is achieved through the use of traditional **send** and **recv** primitives, parameterized almost exactly as other message-passing systems. Several variants are provided for semantics, such as heterogeneous exchange and blocking or nonblocking transfer. A significant proportion of the burden of buffer allocation and management, however, is left to the user. Apart from basic message passing, p4 also offers a variety of global operations, including broadcast, global maxima and minima, and barrier synchronization.

1.4.2 Express

In contrast to the other parallel processing systems described in this section, Express [6] toolkit is a collection of tools that individually address various aspects of concurrent computation. The toolkit is developed and marketed commercially by ParaSoft Corporation, a company that was started by some members of the Caltech concurrent computation project.

The philosophy behind computing with Express is based on beginning with a sequential version of an application and following a recommended development life cycle culminating in a parallel version that is tuned for optimality. Typical development cycles begin with the use of VTOOL, a graphical program that allows the progress of sequential algorithms to be displayed in a dynamic manner. Updates and references to individual data structures can be displayed to explicitly demonstrate algorithm structure and provide the detailed knowledge necessary for parallelization. Related to this program is FTOOL, which provides in-depth analysis of a program including variable use analysis, flow structure, and feedback regarding potential parallelization. FTOOL operates on both sequential and parallel versions of an application. A third tool called ASPAR is then used; this is an automated parallelizer that converts sequential C and Fortran programs for parallel or distributed execution using the Express programming models.

The core of the Express system is a set of libraries for communication, I/O, and parallel graphics. The communication primitives are akin to those found in other message-passing systems and include a variety of global operations and data distribution primitives. Extended I/O routines enable parallel input and output, and a similar set of routines are provided for graphical displays from multiple concurrent processes. Express also contains the NDB tool, a parallel debugger that uses commands based on the popular "dbx" interface.

1.4.3 MPI

The Message Passing Interface (MPI) [7] standard, whose specification was completed in April 1994, is the outcome of a community effort to try to define both the syntax

Copyrighted Material

Downloaded from http://direct.mit.edu/books/monograph/2089/bookpreview-pdf/2430723 by guest on 25 May 2025

and semantics of a core of message-passing library routines that would be useful to a wide range of users and efficiently implementable on a wide range of MPPs. The main advantage of establishing a message-passing standard is portability. One of the goals of developing MPI is to provide MPP vendors with a clearly defined base set of routines that they can implement efficiently or, in some cases, provide hardware support for, thereby enhancing scalability.

MPI is not intended to be a complete and self-contained software infrastructure that can be used for distributed computing. MPI does not include necessities such as process management (the ability to start tasks), (virtual) machine configuration, and support for input and output. As a result, it is anticipated that MPI will be realized as a communications interface layer that will be built upon native facilities of the underlying hardware platform, with the exception of certain data transfer operations that might be implemented at a level close to hardware. This scenario permits the provision of PVM's being ported to MPI to exploit any communication performance a vendor supplies.

1.4.4 The Linda System

Linda [3] is a concurrent programming model that has evolved from a Yale University research project. The primary concept in Linda is that of a "tuple-space", an abstraction via which cooperating processes communicate. This central theme of Linda has been proposed as an alternative paradigm to the two traditional methods of parallel processing: that based on shared memory, and that based on message passing. The tuple-space concept is essentially an abstraction of distributed shared memory, with one important difference (tuple-spaces are associative), and several minor distinctions (destructive and nondestructive reads and different coherency semantics are possible). Applications use the Linda model by embedding explicitly, within cooperating sequential programs, constructs that manipulate (insert/retrieve tuples) the tuple space.

From the application point of view Linda is a set of programming language extensions for facilitating parallel programming. It provides a shared-memory abstraction for process communication without requiring the underlying hardware to physically share memory.

The Linda system usually refers to a specific implementation of software that supports the Linda programming model. System software is provided that establishes and maintains tuple spaces and is used in conjunction with libraries that appropriately interpret and execute Linda primitives. Depending on the environment (shared-memory multiprocessors, message-passing parallel computers, networks of workstations, etc.), the tuple space mechanism is implemented using different techniques and with varying degrees of efficiency. Recently, a new *system* scheme has been proposed, at least nominally related to the Linda project. This scheme, termed "Pirhana" [9], proposes a proactive approach

Copyrighted Material

Downloaded from http://direct.mit.edu/books/monograph/2089/bookpreview-pdf/2430723 by guest on 25 May 2025

to concurrent computing: computational resources (viewed as active agents) seize computational tasks from a well-known location based on availability and suitability. This scheme may be implemented on multiple platforms and manifested as a "Pirhana system" or "Linda-Pirhana system."

Copyrighted Material

Downloaded from http://direct.mit.edu/books/monograph/2089/bookpreview-pdf/2430723 by guest on 25 May 2025

Copyrighted Material

Downloaded from http://direct.mit.edu/books/monograph/2089/bookpreview-pdf/2430723 by guest on 25 May 2025

Bibliography

[1] R. Butler and E. Lusk. Monitors, messages, and clusters: The p4 parallel programming system. Technical Report Preprint MCS-P362-0493, Argonne National Laboratory, Argonne, IL, 1993.

[2] L.E. Cannon. A cellular computer to implement the kalman filter algorithm. Phd thesis, Montana State University, 1969.

[3] Nicholas Carriero and David Gelernter. LINDA in context. *Communications of the ACM*, 32(4):444–458, April 1989.

[4] David Cheriton. VMTP: Versatile Message Transaction Protocol. RFC 1045, Stanford University, February 1988.

[5] J. Wang et. al. LSBATCH: A Distributed Load Sharing Batch System. Csri technical report #286, University of Toronto, April 1993.

[6] J. Flower, A. Kolawa, and S. Bharadwaj. The Express way to distributed processing. *Supercomputing Review*, pages 54–55, May 1991.

[7] Message Passing Interface Forum. Mpi: A message-passing interface standard. Computer Science Dept. Technical Report CS-94-230, University of Tennessee, Knoxville, TN, April 1994. (To appear in the International Journal of Supercomputer Applications, Volume 8, Numbers 3/4, 1994).

[8] G. C. Fox, S. W. Otto, and A. J. G. Hey. Matrix algorithms on a hypercube I: Matrix multiplication. *Parallel Computing*, 4:17–31, 1987.

[9] David Gelernter and David Kaminsky. Supercomputing out of recycled garbage: Preliminary experience with Piranha. In *1992 International Conference on Supercomputing*, pages 417–427. ACM, ACM Press, July 1992.

[10] T. Green and J. Snyder. DQS, A Distributed Queuing System. Scri technical report #92-115, Florida State University, August 1992.

[11] M. Litzkow, M. Livny, and M. Mutka. Condor – A hunter of idle workstations. In *Proceedings of the Eighth Conference on Distributed Computing Systems*, San Jose, California, June 1988.

[12] John K. Ousterhout. *Tcl and the Tk Toolkit*. Addison-Wesley, 1994.

[13] J. Postel. Transmission control protocol. RFC 793, Information Sciences Institute, September 1981.

[14] J. Postel. User datagram protocol. RFC 768, Information Sciences Institute, September 1981.

[15] Daniel A. Reed, Robert D. Olson, Ruth A. Aydt, Tara M. Madhyastha, Thomas Birkett, David W. Jensen, Bobby A. A. Nazief, and Brian K. Totty. Scalable performance environments for parallel systems. In Quentin Stout and Michael Wolfe, editors, *The Sixth Distributed Memory Computing Conference*, pages 562–569. IEEE, IEEE Computer Society Press, April 1991.

[16] Sun Microsystems, Inc. XDR: External Data Representation Standard. RFC 1014, Sun Microsystems, Inc., June 1987.

[17] Louis Turcotte. A survey of software environments for exploiting networked computing resources. Draft report, Mississippi State University, Jackson, Mississippi, January 1993.

Copyrighted Material

Downloaded from http://direct.mit.edu/books/monograph/2089/bookpreview-pdf/2430723 by guest on 25 May 2025

Copyrighted Material

Downloaded from http://direct.mit.edu/books/monograph/2089/bookpreview-pdf/2430723 by guest on 25 May 2025